ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

SLAM VISUAL CON CÁMARAS RGB-D BASADO EN OPTIMIZACIÓN DE GRAFO DE POSES

VISUAL SLAM WITH RGB-D CAMERAS BASED ON POSE GRAPH OPTIMIZATION

Realizado por
David Zúñiga Noël
Tutorizado por
Javier González Jiménez y José Raúl Ruiz Sarmiento
Departamento
Ingeniería de Sistemas y Automática

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: En este trabajo abordamos el problema de localización y mapeo simultáneo (SLAM) utilizando únicamente información obtenida mediante una cámara RGB-D. El objetivo principal es desarrollar un sistema SLAM capaz de estimar la trayectoria completa del sensor y generar una representación 3D consistente del entorno en tiempo real. Para lograr este objetivo, el sistema se basa en un método de estimación del movimiento del sensor a partir de información de profundidad densa y en técnicas de reconocimiento de lugares a partir de características visuales. A partir de estos algoritmos, se extraen restricciones espaciales entre fotogramas cuidadosamente seleccionados. Con estas restricciones espaciales se construye un grafo de poses, empleado para inferir la trayectoria más verosímil. El sistema se ha diseñado para ejecutarse en dos hilos paralelos: uno para el seguimiento y el otro para la construcción de la representación consistente. El sistema se evalúa en conjuntos de datos públicamente accesible, alcanzando una precisión comparable a sistemas de SLAM del estado del arte. Además, el hilo de seguimiento se ejecuta a una frecuencia de 60 Hz en un ordenador portátil de prestaciones modestas. También se realizan pruebas en situaciones más realistas, procesando observaciones adquiridas mientras se movía el sensor por dos entornos de interiores distintos.

Palabras claves: SLAM, RGB-D, tiempo real, grafo de poses, Robótica, Visión por Computador.

Abstract: In this work, we address the Simultaneous Localization And Mapping (SLAM) problem using only an RGB-D camera. The main purpose is to develop a SLAM system capable of estimate the full sensor's trajectory and generate a globally consistent 3D reconstruction of the environment in real time. To achieve this goal, we rely on a dense motion estimation algorithm and on a feature based place recognition technique to derive spatial constraints between selected frames from a sequence. All computed spatial constraints are merged into a graph of poses, used to infer the most likely trajectory. The system is designed to run in two parallel threads: one for tracking and the other for mapping. The resulting system is evaluated on a publicly available benchmark for SLAM systems, reaching an accuracy comparable to state-of-the-art SLAM systems in the estimation of the sensor's trajectory. Moreover, the tracking thread runs at 60 Hz on a modest laptop. Also, we test the system in real settings, processing observations acquired while moving the sensor in two different indoor environments.

Keywords: SLAM, RGB-D, real-time, pose graph, Robotics, Computer Vision.

Contents

Chapter 1

Introducction

Robots have been successfully applied to industrial manufacturing, working in structured environments and releasing us from repetitive tasks. By now, robots are also taking place in our everyday life, working in uncontrolled environments and helping us in tasks that require high-level abilities to be completed. The so-called service robots aim to work in *a priori* unknown environments, with the purpose of improving our quality of life.

For instance, TPR-Robina is a robot that guide tours in a museum, and it is also possible to acquire domestic robots to help us keep our houses clean, like Roomba does, which was designed to vacuum an entire level of a facility (see Figure 1.1).

In order to provide services and perform autonomously, robots usually need a representation of their working environment: for example guiding robots need to know where they are, and home cleaning robots need to know where they have not cleaned yet. These representations

(a) TPR-Robina

(b) Roomba 960

Figure 1.1: TRP-Robina museum guide robot (1.2 m height), by Toyota, acquires a new map whenever the environment changes. Roomba home cleaning robot (0.35 m diameter), by iRobot, builds a map of the working environment so it does not lose track of visited places.

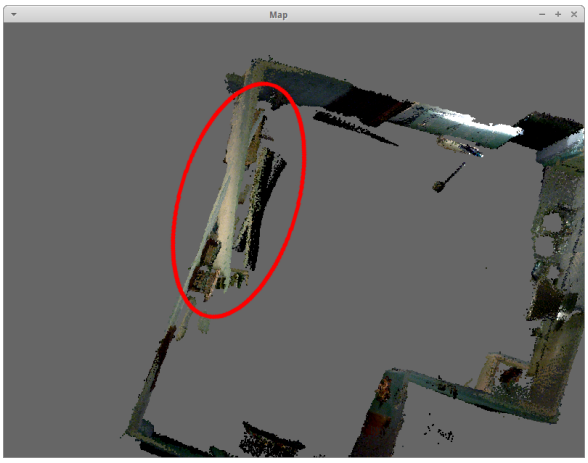


Figure 1.2: The inherent cumulative errors arising from the motion estimation results in inconsistencies when revisiting places.

(commonly referred to as a maps) are required for some important high-level tasks that make a robot truly autonomous, such as planning, navigation or localization. However, a workspace's model is not always available beforehand.

1.1 Motivation

If there is no prior knowledge about the underlying structure of the working environment, and the robot requires a map to operate, it must be acquired online. *Mapping* and *Localization* are two important problems in mobile robotics, and due its correlation they are usually jointly addressed. This leads to the *Simultaneous Localization and Mapping* problem (or SLAM for short) [49].

A SLAM system aims to incrementally build a consistent map of an unknown environment while simultaneously determining the robot's location within this map. Here, by consistent we mean that there are no different representations of a single place (see Figure 1.2).

Following the previous examples, TPR-Robina robot uses SLAM techniques to update its map autonomously according to changes in the museum structure, while Roomba creates a map of visual landmarks using a regular camera for its posterior localization in the house (see Figure 1.3).

Due its complexity, SLAM systems are conceptually divided into two parts: *front-end* and *back-end*. Briefly, the front-end abstracts sensor measurements in a way that can be useful for inferences, performed by the back-end [6].

Visual front-ends rely on cameras to perceive the world, which are relatively cheap sensors that can provide highly distinctive information about the environment [12]. The distinctiveness is required for feature-based motion estimation (visual odometry [8]) and for place recognition (loop closure detection [27]): two main front-end components for robust metric mapping algorithms.

Figure 1.3: SLAM system of Roomba robots based on visual features, by iRobot.

Motion estimation algorithms compute the relative transformation between two consecutive poses, and they are subject to cumulative errors. Place recognition techniques are usually applied to compute additional spatial constraints, independent from odometry ones. These additional constraints allow to bound the cumulative errors and to generate globally consistent representations. The back-end performs this last step: it infers the most likely trajectory taking into account all computed constraints.

For a robot operating in indoor environments, RGB-D cameras are interesting sensors to consider, as they provide color along with depth information for each single pixel. This allows to use traditional computer vision techniques on the RGB images and Depth maps, as well as registration methods working with the point cloud generated from the depth measurements (see Figure 1.4).

1.2 Goals

The main goal of this work is to develop an efficient visual SLAM system that relies on information taken from an RGB-D camera, representing the trajectory followed by the sensor with a graph of poses. The system should be able to run in real-time on a modest laptop and reconstruct 3D dense maps of indoor environments without any prior knowledge of their underlying structure.

This main goal can be divided into the following specific goals:

1. **Measurement representation**: RGB-D cameras provide more than 300,000 depth measurements on a single observation, therefore efficient representations should be used to avoid intractable memory requirements.

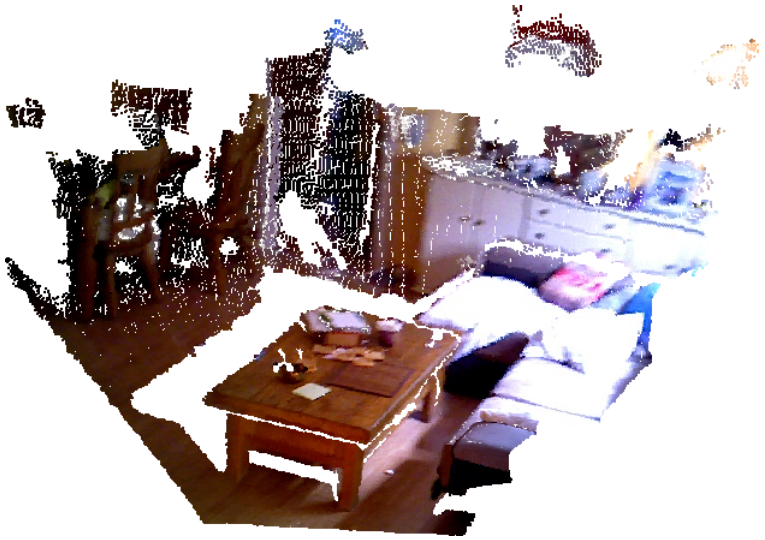


Figure 1.4: A dense 3D point cloud generated from an RGB-D observation. The position in the three-dimensional space of each pixel can be derived from the acquired depth data and the pinhole camera model.

2. **Keyframe selection**: Since these cameras record observations at 30 Hz, efficient representations are not enough. Only carefully selected frames should be considered.

3. **Pose graph modeling**: All spatial constraints from visual odometry and loop closure detection should be integrated into a single model, by building a graph of 3D sensor poses.

4. **Global consistency**: The most likely trajectory should be inferred from all computed constraints, resulting in a globally consistent representation of the environment.

1.3 Related Work

Since the introduction of low-cost RGB-D sensors several SLAM systems have been developed relying on such cameras as the only input device. To the best of our knowledge, the first SLAM system based on this sensors was developed by Henry *et. al.* [17]. Their system extracts visual features along with 3D information to estimate the sensor pose and build a graph of spatial constraints. In order to reduce the number of nodes in the graph, keyframes are selected based on the number of features matched with the previous keyframe. On a separated execution thread, to detect when the sensor is revisiting a place, loop closures are found with previous keyframes following a Bag of Words approach [46]. With this information the pose graph is optimized, resulting in a globally consistent map.

The place recognition method described above tries to detect a loop closure by comparing the last keyframe with each previous one, which results in a linear growth of the time complexity with respect to the number of keyframes. In order to ensure real-time execution for large-scale

environments, the RTAB-MAP system, developed by Labbé and Michaud [25], keeps a fixed number of keyframe candidates for loop closure detection. However, this strategy negatively affects the place recognition performance.

Alternatively, more efficient place recognition methods can be used for long trajectories under real-time constraints. This is the case of DBoW2, proposed by Gálvez-López and Tardós [11], a Bag of Words approach built on top of a binary feature space, allowing to detect loop closures in databases containing 20,000 images at 30 Hz. This approach for place recognition is used, for instance, by the monocular ORB-SLAM system (developed by Mur-Artal *et. al.* [32]) through the utilization of ORB visual features.

Following a different perspective, Kerl *et. al.* [23] propose a dense (i. e. featureless) approach for visual SLAM, based on a fast dense visual odometry method that minimizes both intensity and depth errors from RGB-D observations. Keyframes are selected based on an entropy measure of the covariance of the pose estimation. Loop closure candidates are keyframes that fall in a radius search from the current keyframe 3D position, being validated with the same entropy metric. Although loop closure methods with metric search works well for small environments where loops occur frequently, they can fail in environments with large loops [33].

In this work, we present an hybrid SLAM system that leverages the short execution times of dense visual odometry methods along with the robustness of feature-based place recognition techniques.

1.4 Resources

The SLAM system was written in the C++ programming language, relying on the following open-source libraries:

- **MRPT**: A general library to develop robotic applications.
- **PCL**: A large scale library for 3D point cloud processing.
- **OpenCV**: A library for computer vision and image processing.
- **g2o**: A general framework for optimizing graph-based non-linear functions.
- **DBoW**: A library for binary Bag of Words image representation and retrieval.
- **Boost**: An extensive collection of general-purpose libraries, including smart pointer management and multithread programming features.
- **Eigen**: A library for linear algebra, involving matrices, numerical solvers and algorithms.
- **DLib**: A collection of classes to solve common programming tasks.
- **OpenMP**: An application interface for parallel computing.

as well as on the following hardware components:

- An RGB-D Camera (Asus Xtion Pro Live).
- A Laptop (Compaq CQ58).

1.5 Document outline

The rest of this document is structured as follows:

- **Chapter 2** details the theoretical aspects of the main components behind the developed SLAM system.
- **Chapter 3** covers the experimental evaluation of these components, as well as of the performance of the whole system.
- **Chapter 4** relates the conclusions derived from this work and interesting improvements not covered here but that should be taken into account for future works.

Chapter 2

System Overview

The developed system is composed of a visual front-end and a graph-based back-end (see Figure 2.1). The front-end performs motion estimation with DIFODO [22], a fast dense visual odometry method, and place recognition for both keyframe selection and loop closure detection using DBoW2 [11], an efficient Bag of Words approach relying on FAST [40] features and BRIEF [7] binary descriptor. The back-end performs graph optimization for a globally consistent representation under g2o [24] framework.

The system has been designed to run in two separate threads: one for tracking and keyframe selection, and the other for loop closure detection and pose graph optimization. The tracking thread must process observations at sensor speed in order to achieve real-time performance, while the mapping thread does not have strict run-time constraints and therefore is reserved for algorithms with linear growth complexity. For the parallel execution of the threads we rely on the OpenMP API [37].

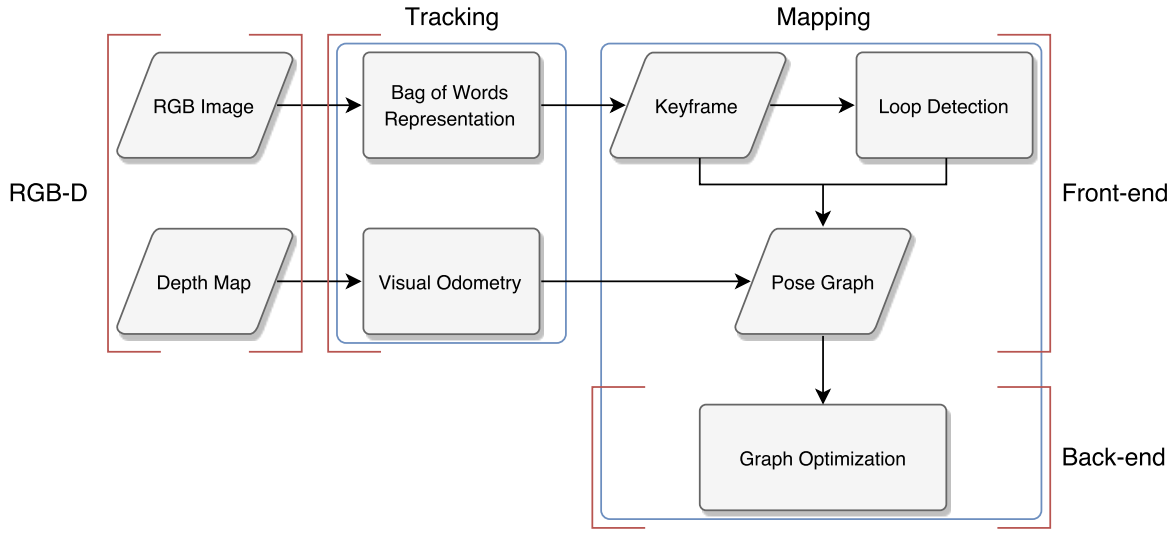


Figure 2.1: Flowchart of the developed SLAM system for a single RGB-D camera. Tracking and mapping threads are executed in parallel on a dual-core processor.

2.1 Front-end

A front-end abstracts sensor measurements for *maximum-a-posteriori* estimation. In this work, we use a single RGB-D camera as sensor, and the input measurements are represented as two functions:

$$I : \Omega \to \mathbb{N} \tag{2.1}$$

$$Z : \Omega \to \mathbb{R} \tag{2.2}$$

where I is the intensity value function and Z is the depth value function for each image pixel in the image domain $\Omega \subset \mathbb{N}^2$ (typically $\Omega = 640 \times 480$). The intensity function is computed from the R, G and, B components following the *Rec. 601* [21]:

$$I(\mathbf{p}) = 0.299R(\mathbf{p}) + 0.587G(\mathbf{p}) + 0.114B(\mathbf{p}) \tag{2.3}$$

The front-end component outputs selected frames represented as graph nodes, i.e. state variables subject to future optimization, and constraints between nodes represented as graph edges. In this work, we will consider only spatial constraints from:

1. Odometry, and
2. Loop Closure

The first ones constrain two consecutive keyframes, while the second ones constrain two keyframes, not necessarily consecutive, observing the same place. These spatial constraints, as well as state variables, are represented as elements in the three-dimensional *Special Euclidean Group* $SE(3)$.

Keyframe Selection

To reduce the number of loop closure candidates and the number of state variables, only a subset of selected frames are considered, namely *keyframes*.

Several techniques for keyframe selection exist, and probably the ones based on visual overlap are the most widely extended. For instance, in [17, 32] the same features are employed for motion estimation are reused for keyframe selection, and whenever the number of tracked features from the last keyframe fall below a given threshold, a new keyframe is added. The key idea behind this approach is that the number of tracked features are progressively fewer as the camera moves, and therefore the density of keyframes is adapted to the camera motion.

In [23], they propose a keyframe selection criterion based directly on the uncertainty of the camera motion estimate, using an entropy measure. This measure encapsulates the uncertainty from the covariance matrix into a scalar value. They add a new keyframe whenever the entropy ratio is below a predefined threshold. The entropy ratio is computed between the uncertainty of the motion from last keyframe and of the first captured motion from the last keyframe (between the last keyframe and the immediately following frame from it).

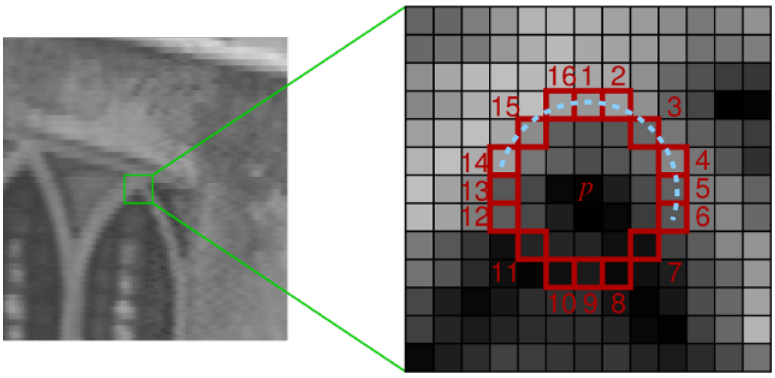


Figure 2.2: A 9 px segment test corner in a circle of 16 px length, by Edward Rosten.

In this work, we follow an appearance-based keyframe selection criterion over a Bag of Words model, trying to negatively affect as less as possible the loop closure detection algorithm. Within this approach, the whole image is described using a global descriptor build from local image feature descriptors. The similarity between two images can be then computed from a simple vector norm.

Bag of Words techniques are taken from text retrieval in document processing, and used in computer vision as object or scene retrieval [46], often called Bag of Visual Words. The general idea consists in building a properly weighted histogram (or a BoW vector) over a finite number of keywords w: the *vocabulary*. In computer vision, instead of keywords, image feature descriptors are used. Each extracted local descriptor is matched to a visual word from the vocabulary to computed the histogram, which acts as global scene descriptor represented as a w-dimensional vector.

Here, we rely on FAST [40] features and BRIEF [7] binary descriptor, as they have shown good performance in scene retrieval [11] and feature matching for small displacements [16], and are orders of magnitude faster than SIFT [26] or SURF [2] methods.

Features from Accelerated Segment Test (FAST) [40] is a high-speed corner detector from the segment test criterion. A pixel candidate $\mathbf{p} \in \Omega$ is considered to be a corner if the Bresenham circle around it contains exactly n consecutive pixels, all brighter or all darker than the candidate pixel (see Figure 2.2). Brighter and darker pixels are defined as:

$$S_{\text{bright}} = \{\mathbf{x} \in \Omega_p \mid I(\mathbf{x}) > I(\mathbf{p}) + t\} \tag{2.4}$$

$$S_{\text{dark}} = \{\mathbf{x} \in \Omega_p \mid I(\mathbf{x}) < I(\mathbf{p}) - t\} \tag{2.5}$$

for a given threshold t, where Ω_p denotes the circle pixels around $\mathbf{p}$. Within this definition, most non-corner pixels can be excluded before completing the full segment test, speeding up the computation time required for the corner detection.

To avoid the detection of multiple adjacent keypoints for the same corner, a non-maximal suppression step is applied to discard redundant detections. For this purpose, an efficient score

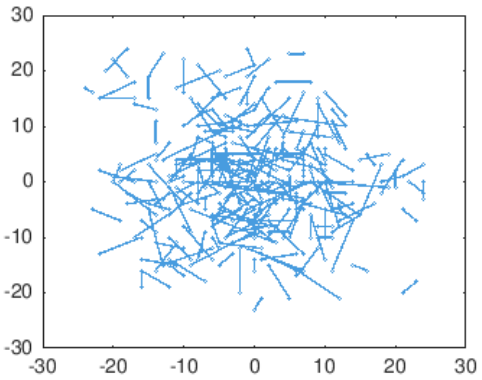


Figure 2.3: Spatial arrangement of the close binary tests used with BRIEF descriptor.

function is computed for each detected feature:

$$V(\mathbf{p}) = \max\Big(\sum_{\mathbf{x}\in S_{\text{bright}}} |I(\mathbf{x}) - I(\mathbf{p})| - t, \sum_{\mathbf{x}\in S_{\text{dark}}} |I(\mathbf{p}) - I(\mathbf{x})| - t \Big) \tag{2.6}$$

keeping only local maxima keypoints.

Binary Robust Independent Elementary Features (BRIEF) [7] is a simple binary descriptor of a local patch around a detected feature. Given a keypoint $\mathbf{p} \in \Omega$, the local descriptor is computed using a intensity difference test over a predefined sequence of sampling points, generating a binary string $\mathbf{b}(\mathbf{p})$ of length L. The test function is defined as

$$\tau(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } I(\mathbf{x}) < I(\mathbf{y}) \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

where $\mathbf{x}$ and $\mathbf{y}$ are a pair of sampling points. The sequence of sampling points is generated randomly from the normal distributions [10]:

$$\mathbf{x}_i \sim \mathcal{N}(\mathbf{0}, \frac{1}{25} S^2) \tag{2.8}$$

$$\mathbf{y}_i \sim \mathcal{N}(\mathbf{x}_i, \frac{4}{625} S^2) \tag{2.9}$$

where S is the side length of the local square patch centered at $\mathbf{p}$.

To reduce the effect of image noise, patches are smoothed using a Gaussian kernel before computing the tests. Finally, the binary string is generated as follows:

$$\mathbf{b}(\mathbf{p}) = \sum_{i=1}^{L} 2^{i-1} \tau(\mathbf{p} + \mathbf{x}_i, \mathbf{p} + \mathbf{y}_i) \tag{2.10}$$

From these features, the vocabulary for the Bag of Words model is generated by a hierarchical quantization of the descriptor space into w words [35]. Training features are partitioned into k clusters by k-means [1] algorithm, and this procedure is repeated for each new cluster up to d levels, yielding a vocabulary tree with $w = k^d$ leaves.

Each word is weighted according to its *inverse document frequency*

$$idf(i) = \log \frac{N}{N_i} \tag{2.11}$$

where N is the number of training images, and N_i the number of occurrences of the word i in these images. This metric downweights very frequent words in the training images, being less discriminative.

To convert an image I (represented by its BRIEF description from FAST features) into a BoW vector $\mathbf{v} \in \mathbb{R}^w$, its binary descriptors traverse the tree (from root to leaves) by selecting at each level the immediate descendant node that minimizes the Hamming distance:

$$d(\mathbf{a}, \mathbf{b}) = \sum_i a_i \oplus b_i \tag{2.12}$$

where $\oplus$ is the XOR binary operator.

In addition, the *term frequency* for each word in the image I is computed as

$$tf(i, I) = \frac{n_i}{n_I} \tag{2.13}$$

where n_i stands for the number of occurrences of word i in image I and n_I for the total number of words in I. This weighting gives more importance to frequent words in a particular image, as they describe it well.

The resulting vector for an image I is

$$\mathbf{v} = \begin{pmatrix} w_1 \\ \vdots \\ w_w \end{pmatrix}$$

where $w_i = tf(i, I)idf(i)$ is the term frequency–inverse document frequency (*tf-idf*) as proposed in [46].

The similarity between two BoW vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ is computed as a L_1-score [11]:

$$s(\mathbf{v}_1, \mathbf{v}_2) = 1 - \frac{1}{2} \left| \frac{\mathbf{v}_1}{|\mathbf{v}_1|} - \frac{\mathbf{v}_2}{|\mathbf{v}_2|} \right| \tag{2.14}$$

whose value lies in $[0, 1]$.

The range in which these scores varies depends on the query image and the visual words it contains. In order to directly compare similarity scores an additional normalization step is performed. The normalized similarity score η between two BoW vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ is defined as [11]:

$$\eta(\mathbf{v}_i, \mathbf{v}_j) = \frac{s(\mathbf{v}_i, \mathbf{v}_j)}{s(\mathbf{v}_i, \mathbf{v}_{i-1})} \tag{2.15}$$

where $s(\mathbf{v}_i, \mathbf{v}_{i-1})$ is an approximation of the maximum expected similarity for $\mathbf{v}_i$: the similarity with the immediate previous frame $\mathbf{v}_{i-1}$.

Within this approach, a frame i is considered to be a keyframe if the normalized similarity score $\eta(\mathbf{v}_i, \mathbf{v}_j)$ with the last keyframe j is less than a given threshold β.

Visual Odometry

The main purpose of a *visual odometry* algorithm is to estimate the relative sensor motion between two consecutive frames relying only on visual input data [34]. The whole trajectory is then calculated incrementally, composing all relative motions and starting from a reference frame (usually, but not necessarly, the first one).

A typical feature-based visual odometry method processes a sequence of images as follows [8]:

1. Feature Detection
2. Feature Matching
3. Motion Estimation

The feature detection step selects distinctive and repeatable interesting points (2D image points or 3D shape points) from the two consecutive observations. The extracted keypoints are then described locally and matched from one image to the other in a high-dimensional space (the descriptor space). Finally, from the set of corresponding points, the sensor motion is computed.

In contrast to that, dense visual odometry methods don't perform any feature extraction or matching step, instead, the motion is estimated by minimizing some general error function. For instance, ICP [3] algorithm (or its variants [42]) aligns two 3D point clouds by corresponding each point from one cloud to the closest in the other cloud and minimizing the Euclidean distance between correspondences. The accuracy of ICP heavily depends on the initial alignment, while, in general, feature-based methods are more robust to large motions. Therefore, ICP-like algorithms are often used to refine a feature-based alignment [17].

An interesting variant of ICP for indoor environments could be Generalized-ICP [44], as it can be formulated with a plane-to-plane error metric, and then it can cope with larger motions. Instead, in this work we use an efficient alternative to GICP: *Differential Odometry* (DIFODO), as it has shown better execution time as well as accuracy [22].

DIFODO algorithm takes a depth map Z as input and computes a motion estimation between consecutive depth observations from the average linear and angular camera velocities during the time interval elapsed (usually 30 Hz). The sensor velocities are derived by applying the range flow constraint equation [47]

$$\dot{Z} = \frac{\partial Z}{\partial t} + \frac{\partial Z}{\partial u}\dot{u} + \frac{\partial Z}{\partial v}\dot{v} + O(\dot{u}, \dot{v}, \Delta t) \tag{2.16}$$

to each pixel $\mathbf{p} = (u, v)$ in Ω, where $\dot{Z}$ represents the depth map derivative with respect to time t, and $\dot{\mathbf{p}} = (\dot{u}, \dot{v})$ the optical flow [19].

The three partial derivatives of Z in Eq. (2.16) can be directly computed from the consecutive depth images. And $\dot{Z}$, $\dot{u}$ and $\dot{v}$ can be expressed in terms of the camera velocities

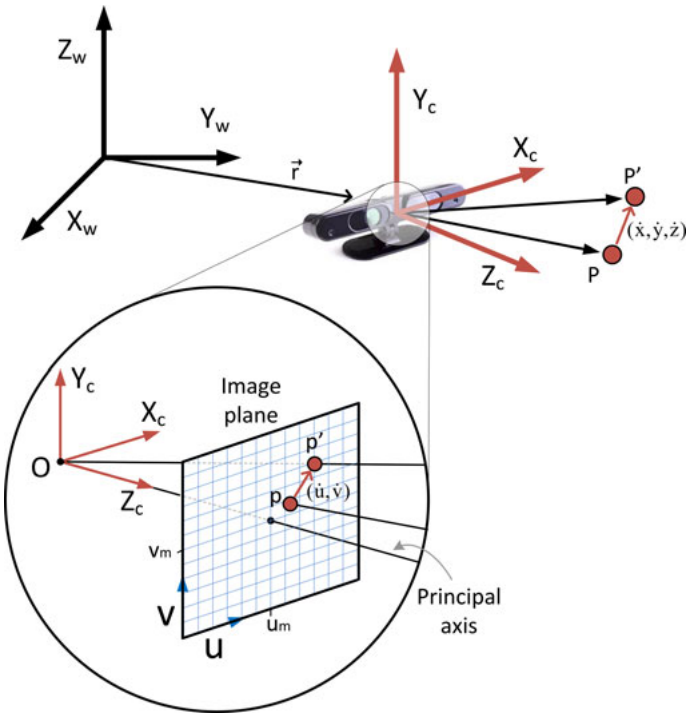


Figure 2.4: Pin-hole camera model, by Mariano Jaimez

$\boldsymbol{\xi} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)^\top$ under a static world assumption:

$$\dot{\mathbf{P}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} -v_x - z\omega_y + y\omega_z \\ -v_y + z\omega_x - x\omega_z \\ -v_z - y\omega_x + x\omega_y \end{pmatrix} \tag{2.17}$$

meaning that every 3D point $\mathbf{P}$ moves with the same velocities as the sensor, but with opposite directions.

Ignoring the higher order terms $O(\dot{u}, \dot{v}, \Delta t)$, Eq. (2.16) becomes

$$\dot{Z} \simeq Z_t + Z_u \dot{u} + Z_v \dot{v} \tag{2.18}$$

where Z_t, Z_u and Z_v are, for simplicity of notation, the partial derivatives of Z. Rearranging terms and replacing the depth map $\dot{Z}$ by the depth coordinate $\dot{z}$ since $Z(u, v) = z$, yields:

$$-Z_t = -\dot{z} + Z_u \dot{u} + Z_v \dot{v} \tag{2.19}$$

From the pin-hole camera model, and assuming that the pixel coordinates of a 3D point $\mathbf{P}$ are time-varying (see Figure 2.4), we get:

$$u = f_x \frac{x}{z} + u_c \Rightarrow \dot{u} = f_x \left(\frac{\dot{x}z - \dot{z}x}{z^2} \right) \tag{2.20}$$

$$v = f_y \frac{y}{z} + v_c \Rightarrow \dot{v} = f_y \left(\frac{\dot{y}z - \dot{z}y}{z^2} \right) \tag{2.21}$$

where (u_c, v_c) is the principal point (image center) and f_x and f_y the focal length values, in pixels.

Expressing the optical flow in terms of $\dot{\mathbf{P}}$ in Eq. (2.19):

$$-Z_t = -\dot{z} + Z_u f_x \Big(\frac{\dot{x}z - \dot{z}x}{z^2}\Big) + Z_v f_y \Big(\frac{\dot{y}z - \dot{z}y}{z^2}\Big) \tag{2.22}$$

and applying the static world assumption yields:

$$\begin{aligned} -Z_t =& \Big(1 + \frac{x f_x}{z^2} Z_u + \frac{y f_y}{z^2} Z_v\Big)(v_z + y\omega_x - x\omega_y) \\ &+ \frac{f_x}{z} Z_u(-v_x + y\omega_z - z\omega_y) + \frac{f_y}{z} Z_v(-v_y - x\omega_z + z\omega_x) \end{aligned} \tag{2.23}$$

The linearization in Eq. (2.18) holds for small motions or if observed points belong to local planar patches, since the higher order terms are negligible. To derive the velocity constraints, a system of linear equations is built from Eq. (2.23). At least, six linearly independent restrictions are required to solve the algebraic system. However, in practice a higher number of points are considered, leading to an over-determined linear system solved by weighted least squares in closed form.

In order to cope with motions larger than a single pixel, a coarse-to-fine scheme is used [5] to compute the optical flow. Within this strategy, a Gaussian pyramid is built by iteratively downsampling and filtering the depth image, allowing to capture larger displacements. The optical flow is solved from coarser to finer levels and at each level, the previous solution is used to warp one image against the other at the same level, leading to image pairs presenting less displacement than the original pair for which the assumption of small motions holds.

Loop Closure Detection

The errors in relative motion estimation that arise from incremental frame-to-frame alignment accumulate over time and eventually yield to significant drift errors. These drifts causes inconsistencies in the map estimation, as a single region may have multiple representations.

A *loop closure* occurs when a previously seen place is revisited, providing additional constraints that can be used to correct the accumulated drift. Therefore, it is necessary to:

1. Detect loop closures between two observations, and
2. Compute the spatial constraint that relates the sensor poses

Most of the approaches to visual loop closure detection are based on the Bag of Visual Words model, but there are other alternatives. For instance, the similarity between two images can be computed from the Locality Sensitive Hashing of the image descriptors [45], avoiding the use of a predefined vocabulary. Instead of local descriptors, a global descriptor can be learned using deep Convolutional Neural Networks to describe the whole image [20]. In this work we use a slightly modified version of the Bag of Visual Words approach, using local binary descriptors [11], that has shown to be more efficient than the other alternatives.

The same Bag of Words model for keyframe selection is reused[1] for loop closure detection, adding an image database of previous observations in order to detect revisited places.

The general idea is to find the best match between the current keyframe and all previous keyframes stored in the database. To speed-up this search, an inverse index is maintained along with the database to retrieve the images that contain a given word. The inverse index allows to compare the current keyframe only against images that share some words in common.

The last K keyframes are excluded from the loop closure candidates as, even having high similarity scores, they don't constitute a true loop closure. A loop closure candidate j is considered for a given keyframe i if the normalized similarity score $\eta(\mathbf{v}_i, \mathbf{v}_j)$ between their BoW vectors $\mathbf{v}_j$ and $\mathbf{v}_i$ is greater than a given threshold α. Only the best scoring candidate is taken into account for future checks.

However, if the expected score $s(\mathbf{v}_i, \mathbf{v}_{i-1})$ is low enough (e.g from fast sensor movements), the normalization step can result in erroneously high similarity values. Therefore, keyframes with low expected scores are discarded from the loop closure detection.

An additional geometric verification of the feature's distribution in the two images is necessary to discard incorrect visually similar loop closure candidates, as introducing false loop closure constraints could lead to even worse errors. In this work, we consider the 3D spatial distribution of the features, as opposed to the 2D distribution proposed in the original approach [11]. A geometric check usually involves the following steps:

1. Feature Matching
2. Outlier Rejection
3. Validation

The feature matching step is only a first guess and it usually contains incorrect matches, which are detected and discarded in the outlier rejection step. Finally, the resulting matches are accepted only if the error of the spatial distribution is acceptable.

The feature matching is performed using the *nearest neighbor distance ratio* [26] policy. A match between a feature from an image and its nearest neighbor in the descriptor space (namely the closest in Hamming distance) on the other image is accepted if the ratio

$$\frac{d(\mathbf{a}, \mathbf{c})}{d(\mathbf{a}, \mathbf{b})} \tag{2.24}$$

is lower than a threshold, where $\mathbf{b}$ and $\mathbf{c}$ correspond to the closest and second-closest matches in one image, respectively, for a feature $\mathbf{a}$ in the other image, and d is the distance function in the descriptor space (the Hamming distance for binary descriptors). This gives us a set of matches $\{\mathbf{a}_i \leftrightarrow \mathbf{b}_i\}$ between local descriptors on both images.

The exact nearest neighbor search has computational complexity of $\Theta(n^2)$ in the number of features, but faster approximations can be used. In this case, the words (or intermediate

[1]Actually, the BoW approach described was originally developed for loop closure detection in [11] and reused here for keyframe selection afterwards, so the BoW model was reused for keyframe selection.

nodes) of the vocabulary reduces the search space for nearest neighbors [11], speeding-up the feature matching process in one hand but sacrificing the exact result in the other hand. For this reason, a direct index is maintained with the database, allowing to retrieve features associated to each word (or tree node) for a given image.

The nearest neighbor distance ratio ensures that the matches are distinctive enough, so that they are less likely to contain incorrect matches. Unfortunately, outliers can be still present, and they negatively affect the quality of the loop closure detection and thus the quality of the overall estimation, so an outlier rejection step is still necessary.

The *RANdom SAmple Consensus* (RANSAC) is a general model fitting paradigm able to cope with a large proportion of outliers. The algorithm takes a minimum size sample from the observed data to compute a model that fits that sample. The support for the computed model is the consensus set: the observed data within an error threshold from the model. These steps are repeated until a reasonably good model (one that has enough support) is computed, or the maximum number of iterations is achieved.

Algorithm 1 Generic RANSAC

1: Select randomly the minimum number of observations required to determine the model parameters.
2: Solve for the parameters of the model.
3: Determine the set of data observations which fit the model within a distance threshold t_r. This is the consensus set of the sample and defines the inliers for the initial data.
4: If the number of inliers is greater than some threshold T, re-estimate the model using all inliers and terminate.
5: Otherwise, repeat steps 1–4 a maximum of N times.

In this case, the observed data is the match set $\{\mathbf{a}_i \leftrightarrow \mathbf{b}_i\}$. For each image, we generate a sparse feature point cloud using the per pixel depth information. This point cloud includes only 3D information for the detected features in the intensity image, giving to each feature descriptor $\mathbf{a}_i$ and $\mathbf{b}_i$ a 3D position $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^3$. For a pixel $\mathbf{p} = (u, v)^\top$, its 3D coordinates $\mathbf{P} = (x, y, z)^\top$ can be derived from the pin-hole camera model:

$$z = Z(u, v) \tag{2.25}$$

$$x = (u - u_c)\frac{z}{f_x} \tag{2.26}$$

$$y = (v - v_c)\frac{z}{f_y} \tag{2.27}$$

where (u_c, v_c) is the principal point (image center) and f_x and f_y the focal length values.

From the corresponding 3D points $\{\mathbf{x}_i \leftrightarrow \mathbf{y}_i\}$, the model is given by a rigid body transformation $T \in SE(3)$ that aligns the two point clouds $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$. A rigid body transformation has a rotation component R and a translation component t:

$$T = \begin{bmatrix} R & t \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{2.28}$$

with $R \in SO(3)$ and $t \in \mathbb{R}^3$. Here, RANSAC performs both outlier rejection and validation for the geometrical verification, but it also computes the spatial constraint required to complete the loop closure detection procedure.

The components R and t are computed as the solution of a least squares formulation [50], minimizing the error function

$$e(T) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{y}_i - T\mathbf{x}_i\|^2 \tag{2.29}$$

in closed form, for a set of n corresponding points.

The rotation component is computed from the *Singular Value Decomposition* (SVD) of the covariance matrix of $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$ and $Y = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$:

$$\Sigma_{xy} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{y}_i - \boldsymbol{\mu}_y)(\mathbf{x}_i - \boldsymbol{\mu}_x)^\top \tag{2.30}$$

where

$$\boldsymbol{\mu}_x = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \tag{2.31}$$

$$\boldsymbol{\mu}_y = \frac{1}{n} \sum_{i=1}^{n} \mathbf{y}_i \tag{2.32}$$

are the mean vectors (or centroids) of X and Y, respectively. Then, $\Sigma_{xy} = UDV^\top$ is the SVD of Σ_{xy} (with $D = diag(d_1, d_2, d_3)$ a diagonal matrix satisfying $d_1 \geq d_2 \geq d_3 \geq 0$).

The optimum rotation R can be computed as

$$R = USV^\top \tag{2.33}$$

when $rank(\Sigma_{xy}) \geq 2$, where

$$S = \begin{cases} I & \text{if } det(\Sigma_{xy}) \geq 0 \\ diag(1, 1, -1) & \text{if } det(\Sigma_{xy}) < 0 \end{cases} \tag{2.34}$$

If $rank(\Sigma_{xy}) = 2$, then

$$S = \begin{cases} I & \text{if } det(U)det(V) = 1 \\ diag(1, 1, -1) & \text{if } det(U)det(V) = -1 \end{cases} \tag{2.35}$$

must be chosen instead of Eq. (2.34).

Finally, the translation t that minimizes the error function is

$$t = \boldsymbol{\mu}_y - R\boldsymbol{\mu}_x \tag{2.36}$$

for the rotation R.

2.2 Back-end

A back-end performs maximum-a-posteriori estimation of the state variables, reasoning about the dependence between these variables. In graph-based SLAM, the state variables are sensor poses, represented as graph nodes, while the dependencies among them are the spatial constraints computed by the front-end, represented as graph edges.

The state vector $\mathbf{x} = (x_1, \cdots, x_n)^\top$ contains the configuration of each node $x_i \in SE(3)$ in the graph. A given constraint $\hat{z}_{ij} \in SE(3)$ relates the variables x_i and x_j.

Pose Graph Optimization

The main goal here is to compute the optimal sensor trajectory that integrates all the constraints computed by the front-end, and thus in a globally consistent representation of the environment.

The original formulation of the SLAM problem as a graph optimization [28] minimizes the spatial distance between corresponding points from different scan matches. It was formulated to work in the two-dimensional space with three degrees of freedom (2 for the position, 1 for the orientation) and for range scans (2D point clouds). Later on, its formulation was extended to work efficiently in the three-dimensional space with six degrees of freedom [4].

In this work, we follow a more general approach [24], taking into account the error between computed transformations instead of between matching points. The error e_{ij} resulting from violating the graph constraint $\hat{z}_{ij}$ is defined as:

$$e_{ij}(\mathbf{x}) = h_{ij} - \hat{z}_{ij} \tag{2.37}$$

where the observation model $h(\mathbf{x})$ predicts a constraint between nodes x_i and x_j, given the state parameters:

$$h_{ij}(\mathbf{x}) = x_i \ominus x_j \tag{2.38}$$

and $\ominus$ is the inverse pose composition.

Here, we consider two types of constraints: odometry and loop closure ones. Odometry constraints form a chain, $\hat{z}_{i(i+1)}$, by applying the observation model to consecutive odometry poses. Loop closure constraints are provided by the loop closure detection module, for some $\hat{z}_{ij}$ with $i > j$ (see Section 2.1).

Therefore, we want the configuration of nodes $\mathbf{x}^*$ that minimizes the overall error[2] $\mathbf{F}(\mathbf{x})$:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) \tag{2.39}$$

$$\mathbf{F}(\mathbf{x}) = \sum_{(i,j)\in\mathcal{C}} e_{ij}(\mathbf{x})^\top \Omega_{ij} e_{ij}(\mathbf{x}) \tag{2.40}$$

where $\mathcal{C}$ is the set of graph constraint indexes and Ω_{ij} represents the information matrix for the constraint $\hat{z}_{ij}$.

[2] Or negative log likelihood [13]

The solution to Eq. (2.39) can be obtained iteratively by approximating the error function by its first order Taylor expansion around a good initial guess $\breve{\mathbf{x}}$:

$$e_{ij}(\breve{\mathbf{x}} + \boldsymbol{\delta}) \simeq e_{ij}(\breve{\mathbf{x}}) + \mathbf{J}_{ij}\boldsymbol{\delta} \tag{2.41}$$

where $\mathbf{J}_{ij}$ is the Jacobian of $e_{ij}(\mathbf{x})$ evaluated at $\breve{\mathbf{x}}$. In the following, we will write $e_{ij} = e_{ij}(\breve{\mathbf{x}})$ for simplicity of notation.

Then, substituting Eq. (2.41) in the error terms of Eq. (2.40):

$$\begin{aligned}
\mathbf{F}(\breve{\mathbf{x}} + \boldsymbol{\delta}) &= \sum_{(i,j)\in\mathcal{C}} e_{ij}(\breve{\mathbf{x}} + \boldsymbol{\delta})^\top \Omega_{ij} e_{ij}(\breve{\mathbf{x}} + \boldsymbol{\delta}) && (2.42)\\
&\simeq \sum_{(i,j)\in\mathcal{C}} (e_{ij} + \mathbf{J}_{ij}\boldsymbol{\delta})^\top \Omega (e_{ij} + \mathbf{J}_{ij}\boldsymbol{\delta}) && (2.43)\\
&= \sum_{(i,j)\in\mathcal{C}} \underbrace{e_{ij}^\top \Omega_{ij} e_{ij}}_{c_ij} + 2\underbrace{e_{ij}^\top \Omega_{ij}\mathbf{J}_{ij}}_{\mathbf{b}_ij}\boldsymbol{\delta} + \boldsymbol{\delta}^\top \underbrace{\mathbf{J}_{ij}^\top \Omega_{ij}\mathbf{J}_{ij}}_{H_{ij}}\boldsymbol{\delta} && (2.44)\\
&= c + 2\mathbf{b}^\top\boldsymbol{\delta} + \boldsymbol{\delta}^\top H \boldsymbol{\delta} && (2.45)
\end{aligned}$$

where $c = \sum c_{ij}$, $\mathbf{b} = \sum \mathbf{b}_{ij}$ and $H = \sum H_{ij}$.

The overall graph error can be minimized in $\boldsymbol{\delta}$ by minimizing the quadratic equation Eq. (2.45), i.e. solving the linear system

$$H\boldsymbol{\delta}^* = -\mathbf{b} \tag{2.46}$$

The solution for a single iteration is obtained by adding $\boldsymbol{\delta}^*$ to the initial guess:

$$\mathbf{x}^* = \breve{\mathbf{x}} + \boldsymbol{\delta}^* \tag{2.47}$$

The following iteration takes $\mathbf{x}^*$ as the initial guess, and the whole process is repeated until some convergence condition holds. This is the standard Gauss-Newton non-linear least squares optimization algorithm, which works only for state variables that span over an Euclidean space. The general method can be applied to the minimal parameterization for 3D poses, as it is in $\mathbb{R}^6$ (6 degrees of freedom, 3 for translation and 3 Euler angles for rotation).

However, the minimal parameterization is subject to singularities (namely, gimbal lock), that can be overcome using an over-parameterized representation. The parameterization in $SE(3)$ has the manifold structure $SO(3)\times\mathbb{R}^3$ and, clearly, the rotational component span over a non-Euclidean space. In this case, the standard minimization algorithm is not applicable, since Eq. (2.47) can break the orthogonality constraint induced by the over-parameterized representation for the rotations, the *Special Orthogonal Group* $SO(3)$.

An alternative is to express the increments δ_i in a space different from the one for state variables x_i. In this case, we want over-parameterized representations for the state variables but a minimal representation for the perturbation term, as for small increments it is far from singularities.

In order to use the standard optimization algorithms with different parameterizations, and to apply Eq. (2.47) and (2.37), two new operators are needed. Let $\mathcal{S} = SO(3) \times \mathbb{R}^3$ be the manifold structure for $SE(3)$. Then the encapsulation operator

$$\boxplus : \mathcal{S} \times \mathbb{R}^6 \to \mathcal{S} \tag{2.48}$$

can replace the vector sum in Eq. (2.47), yielding:

$$\mathbf{x}^* = \breve{\mathbf{x}} \boxplus \boldsymbol{\delta}^* \tag{2.49}$$

where $\mathbf{x} \boxplus \boldsymbol{\delta} = (x_1 \boxplus \delta_1, \cdots, x_n \boxplus \delta_n)^\top$ with $x_i \in SE(3)$ and $\delta_i \in \mathbb{R}^6$. This operator applies a small increment parameterized over an Euclidean space to a manifold state variable.

The inverse encapsulation operator

$$\boxminus : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^6 \tag{2.50}$$

replaces the vector difference in Eq. (2.37):

$$e_{ij}(\mathbf{x}) = h_{ij} \boxminus \hat{z}_{ij} \tag{2.51}$$

Finally, the first order approximation for the error function is then

$$e_{ij}(\breve{\mathbf{x}} \boxplus \boldsymbol{\delta}) \simeq e_{ij}(\breve{\mathbf{x}}) + \left.\frac{\partial(e_{ij}(\breve{\mathbf{x}} \boxplus \boldsymbol{\delta}))}{\partial \boldsymbol{\delta}}\right|_{\boldsymbol{\delta}=\mathbf{0}} \boldsymbol{\delta} \tag{2.52}$$

The Gauss-Newton algorithm, however, is not guaranteed to converge if the initial guess is not close enough to the minimum. The Levenberg-Marquardt algorithm improves the convergence by introducing a control parameter λ in Eq. (2.46), behaving like a gradient descent when the current solution is far from the minimum and like Gauss-Newton when approaching the minimum. With this control parameter, Eq. (2.46) becomes

$$(H + \lambda I)\boldsymbol{\delta}^* = -\mathbf{b} \tag{2.53}$$

For more information and the mathematical details, we refer the reader to [13, 18, 29].

Chapter 3

Experimental Evaluation

Our main concerns regarding SLAM components are their execution time and accuracy. The methods are evaluated on an Intel® Core™ i3-2328M CPU at 2.20 GHz with a shared 4GB SO-DIMM DDR3 RAM at 1333 MHz, while they are compared with ground-truth reference information for a quantitative performance measure.

For that purpose, we use the publicly available RGB-D TUM Dataset [48], as it contains sequences for the evaluation of SLAM systems with highly accurate ground truth camera poses. Within this dataset, we choose sequences from the *Handheld SLAM* category, which includes camera motions in office and house-like environments with six degrees of freedom.

More precisely, we use the sequences:

- '`freiburg1_desk`' (*fr1/desk*)
- '`freiburg1_desk2`' (*fr1/desk2*)
- '`freiburg1_room`' (*fr1/room*)
- '`freiburg2_desk`' (*fr2/desk*)
- '`freiburg3_long_office_household`' (*fr3/office*)

being widely used to evaluate RGB-D state-of-the-art SLAM systems.

The accuracy of the whole system is compared with state-of-the-art RGB-D SLAM systems, and, finally, we perform real experiments with a handheld camera. For these live tests, we use an Asus Xtion Pro Live RGB-D camera to acquire 3D observations in real-time, moving the sensor around our lab and in a typical living room.

Unless otherwise explicitly stated, we use the default parameters for each algorithm. In all cases, we rely on open-source implementations, so the default parameters can be easily found.

Table 3.1: Execution time for DIFODO with different resolutions.

Resolution	Execution time (ms)			
	Mean	Std	Min	Max
640×480	88.06	8.48	51.91	143.18
320×240	21.49	1.99	12.75	37.85
160×120	5.56	0.50	3.51	14.25

3.1 Visual Odometry

Regarding the visual odometry algorithm, we are interested in how the resolution of the finer level considered by DIFODO affects its performance and execution time, in our own hardware settings[1]. For these experiments we run the algorithm as implemented in the MRPT library [31].

We start by evaluating how the downsampling factor affects the execution time of the algorithm, with the maximum number coarse-to-fine levels allowed for each resolution. We run DIFODO in the five sequences with three different resolutions: 640×480, 320×240 and 160×120. In Table 3.1, we summarize the average execution time, as well as standard deviation, minimum and maximum for all executions (a total of 7777 executions, for each resolution).

From these results, as one may expect, the lowest resolution is the fastest in execution time. Moreover, it is the only that ensures a real-time execution, running even faster than 60 Hz. But this high speed was achieved by ignoring information in the downsampling step, and these strategies often negatively affects the accuracy in the estimation.

To measure the accuracy, the *relative pose error* (RPE) [48] metric is used, where the estimated trajectory $P_1, \ldots, P_n \in SE(3)$ and ground-truth $Q_1, \ldots, Q_n \in SE(3)$ are compared over a time interval Δ:

$$E_i = (Q_i^{-1} Q_{i+\Delta})^{-1} (P_i^{-1} P_{i+\Delta}) \tag{3.1}$$

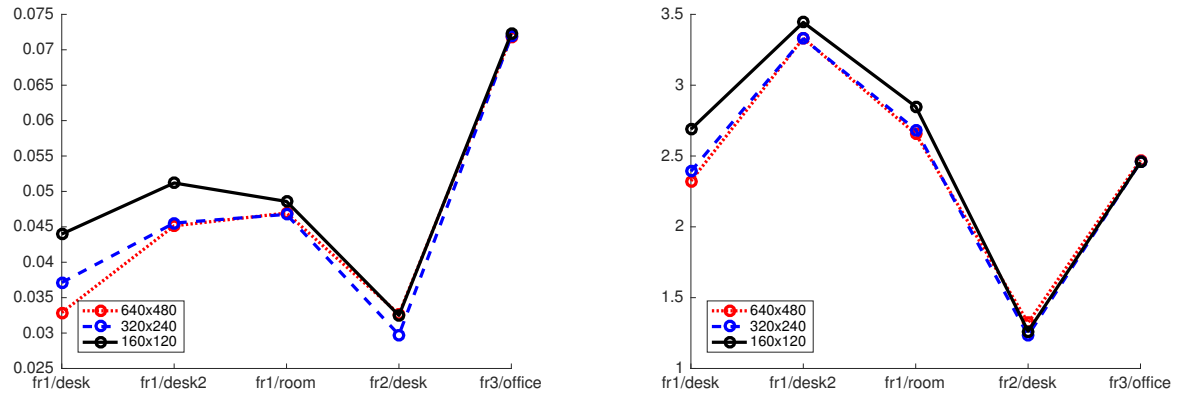
for $i = 1, \ldots, m$, where $m = n - \Delta$.

For the sake of simplicity, we assumed that both sequences were time-synchronized, equally sampled and with the same length, as well as a fixed time interval. In practice they have different start times and sampling rates, so additional data association and interpolation steps are performed.

From the relative pose errors, the root mean squared error (RMSE) is computed for translational and rotational errors:

$$RMSE_t(E_{1:m}, \Delta) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \|trans(E_i)\|^2} \tag{3.2}$$

[1]Also, we run the latest version of DIFODO, so the result can vary from the ones presented in the original publication.



(a) Translational

(b) Rotational

Figure 3.1: Translational and rotational RMSE, in m/s and deg/s, respectively.

$$RMSE_r(E_{1:m}, \Delta) = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \|ang(E_i)\|^2} \tag{3.3}$$

where $trans(E_i) \in \mathbb{R}^3$ represents the translational component and $ang(E_i) \in \mathbb{R}$ is computed from the rotational component $R_i \in SO(3)$ as follows:

$$ang(E_i) = \cos^{-1}\left(\frac{trace(R_i) - 1}{2}\right) \tag{3.4}$$

The RMSE is influenced by occasionally large errors in the estimates, and therefore a low RMSE drift value indicates a continuously high tracking accuracy. We evaluate the performance of the three considered resolutions of DIFODO in the five sequences, presenting the results in Figure 3.1.

Following the intuition, the highest resolution achieves, in general, the best performance. But the fact is that the three resolutions are close performers.

From these results, we choose to use the lowest resolution (160×120), meaning a downsampling factor of 4, and considering 5 coars-to-fine levels, as it runs at 60 Hz with a high accuracy. For the sake of completeness, in Table 3.2 the translational and rotational RMSE for these parameters are presented. As noted by the original authors [22], the accuracy of the method is more susceptible to measurement errors produced by the sensor in slower motions. Therefore, we evaluate the improvements achieved with this strategy by skipping consecutive frames to simulate faster motions.

In most cases, by skipping a single frame we achieve more accurate results in these sequences. The best improvements for the sequences `fr2/desk` and `fr3/office` are achieved by skipping two consecutive frames. This can be explained from the fact that they were recorded by moving the sensor slowly

Table 3.2: Performance of DIFODO, in terms of translational and rotational RMSE, for the final parameters and skipping fs consecutive frames.

Sequence	Translational (m/s)			Rotational (deg/s)		
	$fs = 0$	$fs = 1$	$Fs = 2$	$Fs = 0$	$fs = 1$	$fs = 2$
`fr1/desk`	0.044	**0.043**	0.078	**2.724**	3.130	6.034
`fr1/desk2`	0.051	**0.047**	0.162	3.454	**3.359**	3.761
`fr1/room`	0.049	**0.047**	0.052	2.787	**2.679**	2.680
`fr2/desk`	0.032	0.025	**0.023**	1.240	0.987	**0.906**
`fr3/office`	0.072	0.019	**0.018**	2.439	1.065	**0.969**

3.2 Loop Closure Detection

For the loop closure detection module, we are interested in the number of correct detections and how this number is affected by the keyframe selection. We also analyze the effect of reducing the sequence length (by applying a keyframe selection crietrion) on the execution time of the loop closure detection algorithm.

We run the implementation of FAST from the OpenCV library [36], while for BRIEF, we run the implementation in DLib [15]. For the binary Bag of Words, we rely on the implementation from DBoW2 []. We use the vocabulary generated in [11], available for download[2], with $w = 10^6$ words from the 'Bovisa_2008-09-01' session of the Rawseeds project[3]. We implemented our own RANSAC procedure, based on the SVD transformation estimation implemented in the PCL [38]. We set the maximum distance threshold to 1 cm, the minimum size consensus set to 12 and a maximum of 500 iterations.

Loop closure detections can be represented as a binary relation between sequence indices. Here, $R(i, j)$ means that a loop between frames (or keyframes, depending on the case) was detected. The precision and recall metrics, to compare the estimated loop closure relation R with a ground truth loop closure relation S, are defined as[4]:

$$precision(R, S) = \frac{|\Gamma_{R \cap S}|}{|\Gamma_R|} \tag{3.5}$$

$$recall(R, S) = \frac{|\Gamma_{R \cap S}|}{|\Gamma_S|} \tag{3.6}$$

where

$$\Gamma_A = \{i \mid i > j + k,\ (i, j) \in A\} \tag{3.7}$$

for a given set A and the discarding parameter K, used to reject recent frames from the loop closure candidates.

[2] `http://doriangalvez.com/resources/DLoopDetector/resources.tar.gz`

[3] `http://www.rawseeds.org/rs/capture_sessions/view/10`

[4] Here we give a formal definition of the precision and recall metrics, as described in [11].

The precision metric is defined as the ratio between the number of correct detections and all the detections fired by the algorithm, and it measures the quality of the detections. For a robust SLAM system, a 100% precision is required, as usually loop closures are unmovable decisions. The recall is defined as the ratio between the correct detections and all loops in the ground truth, and measures the ability of the algorithm to capture the loop closures occurring in a sequence. A higher recall is preferable, as more correction constraints will be available, resulting in a better trajectory estimation.

Since there is no loop closure ground-truth information on the dataset used, we derive this information from ground-truth pose information to evaluate the precision of the loop closure detection method. Following a similar approach as in [43], a loop closure detection is verified (or correct) if:

1. $\|trans(T_{ij})\| < 2\,\text{m}$, and
2. $|ang(T_{ij})| < 45\,\text{deg}$

where $T_{ij} = Q_i^{-1} Q_j$ represents the relative ground-truth transformation between frames number $i, j \in \{1, \ldots, n\}$ in the sequence.

Detected loop closures but not automatically verified from the ground-truth information (meaning at least one of the above conditions does not hold) are manually verified, as it is hard to define a procedure sound and complete for this purpose. With these simple and easy implementable conditions most loop closures are automatically verified, so only a few need human supervision.

Instead of the true recall metric, we use a relative lower bound recall metric, dividing by the sequence length n, or the number of keyframes considered, if it is the case:

$$recall(R, S) = \frac{|\Gamma_{R \cap S}|}{n} \tag{3.8}$$

This definition of recall allows us to measure the trade off between the number of correct loop closures detected and the reduction in the length of the sequence.

The number K of frames discarded from loop closure candidates affects the recall, and depends on the speed of the camera motion. When no keyframes are selected (i.e. all frames are processed), we set $K = 100$ (a bit more than 3 s for sequences recorded at 30 Hz). For keyframe selection criteria, we set $K = 40$, as the expected reduction factor of the frames to process is around 2.5. In order to evaluate the accuracy of the loop closure detection with these values for K for all sequences, we skip two consecutive frames in the slower sequences (namely `fr2/desk` and `fr3/office`) to simulate faster motions.

Table 3.3 shows the results form these experiments. In most cases, using a keyframe selection criterion improves the relative lower bound recall metric, meaning that loop closures are preserved to some degree while reducing the total number of frames to process. Figure 3.2 shows the effect of reducing the sequence length in the maximum execution time spent by the loop closure detection algorithm. The maximum execution time is reduced roughly by

Table 3.3: Relative lower bound recall performance of the Loop Closure Detection algorithm for different values of the similarity threshold β.

Similarity	Sequence				
	`fr1/desk`	`fr1/desk2`	`fr1/room`	`fr2/desk`	`fr3/office`
No Keyframe	1.79%	2.13%	0.68%	1.28%	**4.46%**
$\beta = 0.8$	**2.70%**	1.59%	**1.34%**	1.33%	3.88%
$\beta = 0.7$	1.91%	2.67%	1.32%	**2.17%**	3.44%
$\beta = 0.6$	1.09%	**3.05%**	0.76%	1.05%	3.07%

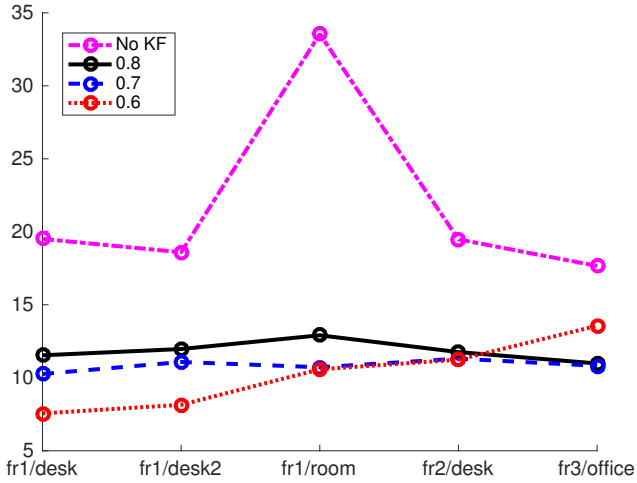


Figure 3.2: Maximum execution time spend on each sequence, for various keyframe selection criteria, in ms. 'No KF' means all frames were processed.

the same factor as number of frames to process, since the computational complexity for loop closure detection grows linearly with the number of frames in the worst case.

For the subsequent experiments, we set $\beta = 0.8$, has it has shown the best performance for the relative lower bound recall metric and runs at 60 Hz for a sequence with more than 500 keyframes.

3.3 The Whole System

In this section, we compare our system with state-of-the-art SLAM systems, in terms of the *Absolute Trajectory Error* (ATE) [48] metric. We also evaluate the improvement in the trajectory estimation achieved by optimizing the graph of poses, and how this procedure is affected by the keyframe selection criterion, in execution time and in terms of the ATE metric.

This metric measures the accuracy of the resulting estimated trajectory against the ground-truth trajectory, taking into account the global consistency. Each keyframe pose $P_i \in SE(3)$

is assigned to a corresponding ground-truth pose $Q_i \in SE(3)$ based on the timestamp values. The 3D corresponding points, obtained from the translation component of each pose, are aligned by the rigid body transformation $S \in SE(3)$, from a least squares solution. The error between corresponding poses is then

$$F_i = Q_i^{-1} S P_i \tag{3.9}$$

From these errors, the RMSE is computed as

$$RMSE(F_{1:n}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|trans(F_i)\|^2} \tag{3.10}$$

for the translational component of F_i.

Within this metric, we compare our approach with different SLAM systems. More precisely, we consider the DVO-SLAM [23] and RGB-D SLAM [9] systems as they minimize both photometric and geometric errors form RGB-D observations. The former is based only on dense methods, while the latter uses feature-based ones. We also compare it with systems that minimizes only the geometric error following a dense formulation. In this class fall the Kintinuous [51] system: an extension to large scale environments of KinectFusion [39] algorithm, and the CPA-SLAM [30]: a very recent system based on plane segmentation of depth information.

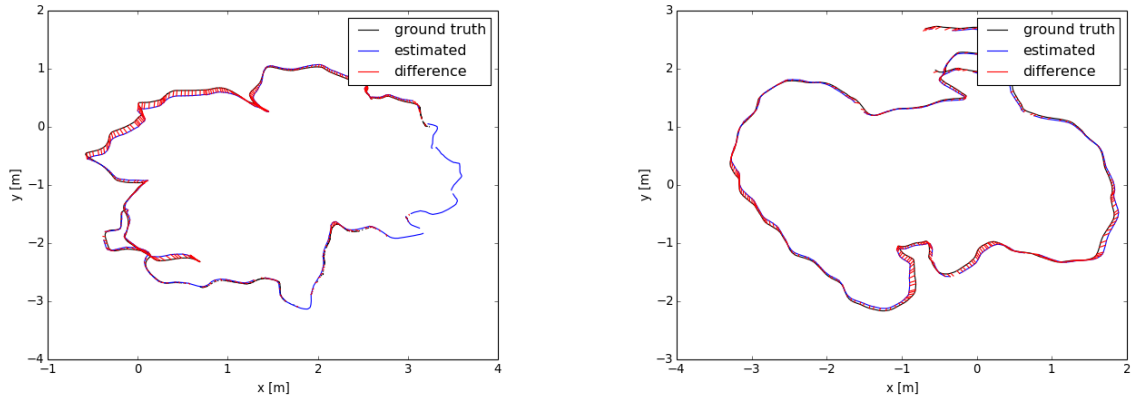
The accuracy of the visual odometry algorithm directly affects the accuracy of the whole system. For this reason, we run our system for several frame skip values, as it has shown to be a reliable strategy to improve the trjectory estimation. The results from these experiments are presented in Table 3.4.[5] Despite the low resolution used and the simplicity of the feature extraction algorithms, our system reaches an accuracy comparable to state-of-the-art SLAM systems (see Figure 3.3).

The execution time of the whole system for all sequences is shown in Table 3.5. The tracking thread (visual odometry and keyframe selection) runs at 60 Hz on a single CPU. This performance, for the best of our knowledge, has never been reported in the literature.

For the analysis of the back-end module, we summarize in Table 3.6 the improvements achieved in terms of absolute trajectory RMSE in contrast to a visual odometry only trajectory (i.e. without any optimization), along with general information about the number of keyframes and loop closures considered for each sequence, setting $fs = 1$. In all cases, as one may expect, there is a great improvement in the estimated trajectory.

In Figure 3.4 we show the effect of the keyframe selection strategy on the maximum execution time of the graph optimization and on the trajectory estimation, for each sequence. Has the intuition says, the execution time is improved in all cases, while, in general, the error in the estimated trajectory increases when only keyframes are considered. This is mostly due

[5]The errors for each system were taken form their original publications, as well as form [14]. If there is any contradiction between these values, we take always the smallest one.



(a) `fr2_desk` (b) `fr3_office`

Figure 3.3: The ground-truth and estimated trajectories aligned, for the longest sequences. The absolute trajectory error is computed from these differences.

Table 3.4: Absolute Trajectory Error RMSE (m) comparison with state of the art approaches.

Frame skip	Sequence				
	`fr1/desk`	`fr1/desk2`	`fr1/room`	`fr2/desk`	`fr3/office`
$fs = 0$	0.047	**0.048**	0.068	0.118	0.116
$fs = 1$	**0.039**	0.049	**0.062**	0.064	0.063
$fs = 2$	0.214	0.205	0.082	**0.056**	**0.055**
CPA-SLAM [30]	**0.018**	**0.029**	0.055	0.046	**0.025**
Kintinuous [51]	0.037	0.071	0.075	0.034	0.030
RGB-D SLAM [9]	0.023	0.043	0.084	0.057	0.032
DVO-SLAM [23]	0.021	0.046	**0.053**	**0.017**	0.035

to a lower number of loop closures. The execution time was plotted using a logarithmic scale, and it is improved in a greater factor than the additional error induced by a keyframe selection strategy.

Finally, we tested the developed system in real-life scenarios, by and RGB-D camera on indoor environments. We run it the MAPIR [6] and in a typical living room. The resulting 3D maps are shown in Figure 3.5. In both cases, the system was able to detect loop closures and to correct the accumulated drift, generating consistent representations. For larger environments with few loop closures, the accuracy of the representation can be highly affected, yielding to unacceptable errors.

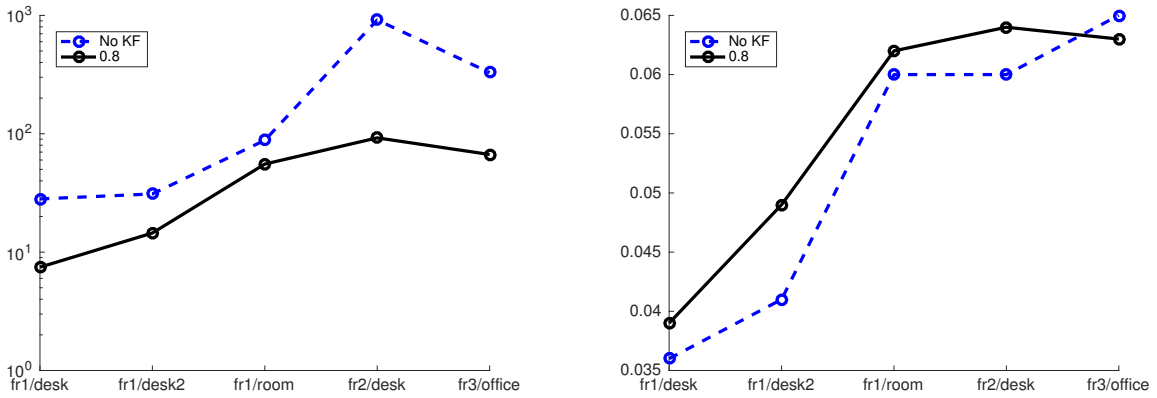
[6] `http://mapir.isa.uma.es/mapirwebsite/`

Table 3.5: Execution time for all components of the SLAM system.

Component	Execution time (ms)			
	Mean	Std	Min	Max
DIFODO	5.80	0.55	0.98	13.82
FAST	1.95	0.51	0.34	5.61
BRIEF	3.12	0.33	2.07	7.34
Keyframe Selection	4.32	0.39	1.76	9.18
Loop Closure Detection	5.23	0.96	2.21	20.09
Graph Optimization	106.11	57.30	17.54	196.31

Table 3.6: Graph optimization vs no graph... .

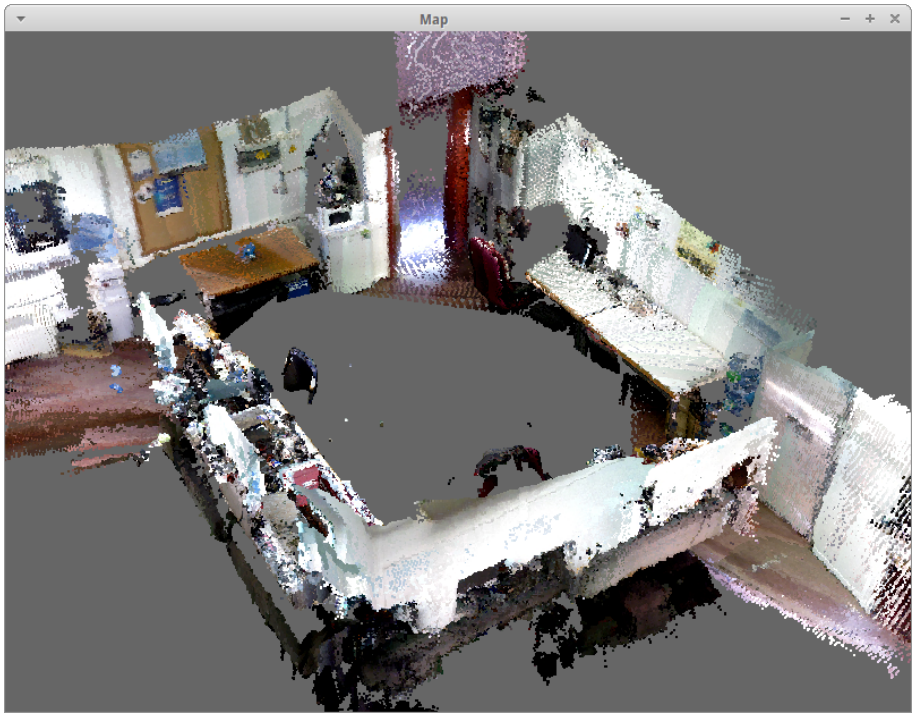
Sequence	Opt.	w/o Opt.	Frames	Keyframes	Loop Closures
`fr1/desk`	0.039	0.113	559	120	4
`fr1/desk2`	0.049	0.078	609	131	1
`fr1/room`	0.062	0.159	1310	284	1
`fr2/desk`	0.064	0.310	2182	536	11
`fr3/office`	0.063	0.322	2488	425	17



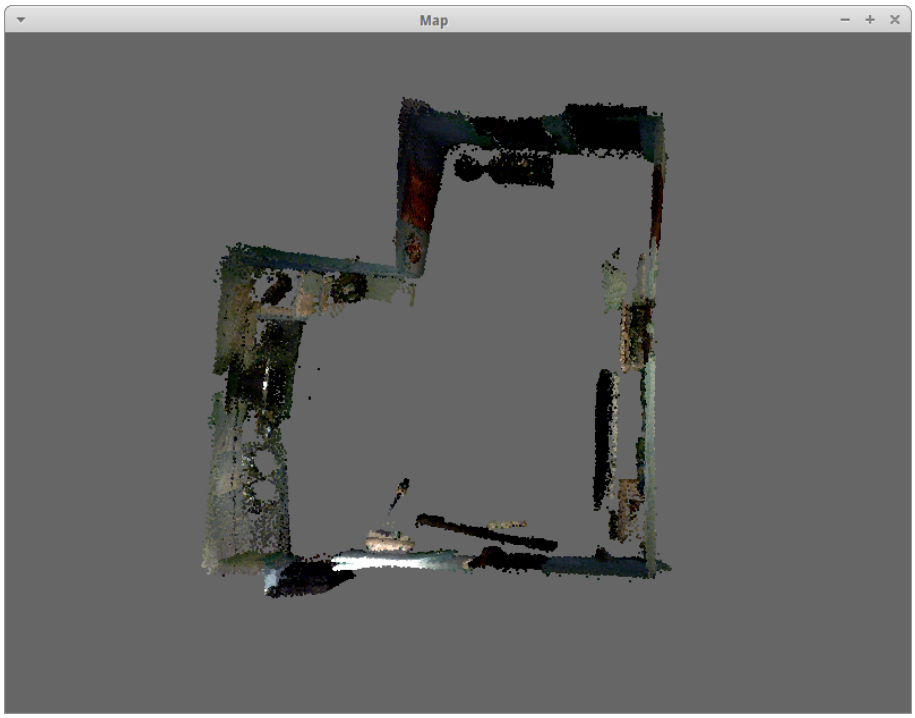
(a) Execution time

(b) Trajectory Error

Figure 3.4: Maximum execution time and absolute trajectory RMSE, in ms and m, respectively. Note that the execution time is plotted on a logarithmic scale.



(a) Lab



(b) Living room

Figure 3.5: Globally consistent 3D dense representations of two different indoor environments.

Chapter 4

Conclusions and Future Work

In this work, we developed a SLAM system that relies solely on information taken from an RGB-D camera. The system is able to estimate the full trajectory of the sensor and incrementally build a consistent 3D representation for small indoor environments. Real-time execution is achieved by dividing the execution into two threads: tracking and mapping.

The tracking thread, composed of the visual odometry and keyframe selection components, runs at 60 Hz on a modest laptop. The mapping thread runs the remaining components, with higher computational demands.

The accuracy of the estimated trajectory by our SLAM system was evaluated on a publicly available benchmark for SLAM systems, and performed comparable to state-of-the-art RGB-D SLAM systems.

The specific goals considered for this work were addressed as follows:

- **Measurement representation**: RGB-D observations are represented by sparse feature point clouds, using the FAST detector.
- **Keyframe selection**: Interesting frames are selected based on visual similarity, computed with a binary Bag of Words (DBoW2) over BRIEF descriptions.
- **Pose graph modeling**: Visual odometry constraints are computed from DIFODO poses estimation, while loop closure are detected based on the same Bag of Words as for keyframe selection and the spatial constraints are computed from 3D point correspondences.
- **Global consistency**: Trajectory poses are estimated from all computed constraints, under the g2o framework.

Even having accomplished the goals we set for this work, there is still room for improvements. Regarding the keyframe selection algorithm, an interesting alternative can be thresholding the entropy measure [23] of the covariance matrix of the motion estimation computed by DIFODO.

As we see in the experimental evaluation, the accuracy in the trajectory estimated by DIFODO can be improved by skipping consecutive frames. The number of frames depends on the speed at which the sensor is moving, and an optimal value could be derived online, from the sensor velocities computed during the estimation.

For the loop closure detection module, we only consider the best match for the geometric validation procedure. A higher number of candidates can be considered, depending on the computational load, to increase the detection recall. These candidates can be sorted by the uncertainty of their pose, favouring candidates with lower pose uncertainty. Also, more robust features and descriptors can be used, such as ORB [41].

Finally, the information matrix in the graph optimization formulation should be used to allow higher variations in poses with higher uncertainty, in order to preserve poses with lower uncertainty as much as possible.

Conclusiones y Trabajos Futuros

En este trabajo se presenta un sistema SLAM que depende únicamente de información obtenida mediante una cámara RGB-D. El sistema es capaz de estimar la trayectoria completa del sensor y construir de manera incremental una representación 3D consistente de entornos de interiores pequeños. Se alcanza la ejecución en tiempo real dividiéndola en dos hilos: uno para el seguimiento y el otro para la construcción de la representación consistente.

El hilo de seguimiento, compuesto por los algoritmos de odometría visual y de selección de fotogramas a procesar, alcanza los 60 Hz en un ordenador portátil de prestaciones modestas. El otro se encarga de la ejecución el resto de componentes, que tienen un crecimiento lineal en cuanto a complejidad computacional.

La precisión de la trayectoria estimada por este sistema ha sido evaluada en conjuntos de datos públicamente accesibles, obteniendo resultados comparables a sistemas de SLAM del estado del arte.

Los objetivos específicos propuestos para este trabajo han sido abordados del siguiente modo:

- **Representación**: Las observaciones RGB-D se representan con nubes de puntos dispersas, a partir de puntos de interés obtenidos mediante el detector FAST.

- **Selección de fotogramas**: Los fotogramas relevantes se seleccionan en base a una métrica de similitud visual, calculada mediante una Bolsa de Palabras binarias (DBoW2) sobre descripciones BRIEF.

- **Grafo de poses**: Las restricciones espaciales entre los fotogramas consecutivos se calculan utilizando el algoritmo DIFODO, mientras que las detecciones de cierre de bucle se realizan utilizando la misma Bolsa de Palabras empleada para la selección de fotogramas y las restricciones espaciales entre las detecciones se calculan a partir de correspondencias entre puntos 3D.

- **Consistencia global**: La trayectoria resultante de integrar todas las restricciones calculas se realiza en el marco de la librería g2o.

Aún habiendo cumplido los requisitos establecidos para este trabajo, siguen habiendo mejoras por realizar. Respecto a la selección fotogramas, una alternativa interesante puede ser

utilizar una métrica sobre la entropía [23] de la matriz de covarianza en la estimación de la trayectoria realizada por DIFODO.

Como se puede observar en la evaluación experimental, la precisión en la estimación de la trayectoria realizada por DIFODO se puede mejorar descartando fotogramas consecutivos. El número de fotogramas a descartar depende de la velocidad con la que se mueve el sensor por el entorno, y el número óptimo se podría calcular dinámicamente a partir de la velocidad estimada por el propio algoritmo.

Para la detección de cierre de bucle se ha considerado únicamente el fotograma con mayor similitud para la posterior etapa de verificación geométrica. Se puede considerar un número mayor de candidatos dependiendo de la carga computacional del sistema, con el objetivo de aumentar el número de detecciones. Estos candidatos se podrían ordenar con respecto a la incertidumbre en la estimación de su posición en el espacio, favoreciendo aquellos que tengan una incertidumbre menor. También se pueden usar otros descriptores más robustos, como es el caso de ORB [41].

Finalmente, la matriz de información empleada en la formulación de la optimización de grafos debería utilizarse para permitir mayores variaciones en la estimación de poses con mayor incertidumbre, de modo que se mantengan, cuanto sea posible, las poses con menos incertidumbre.

Bibliography

[1] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[2] H. Bay, T. Tuytelaars, and L. Van Gool. *SURF: Speeded Up Robust Features*, pages 404–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[3] P. J. Besl and H. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992.

[4] D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter, and J. Hertzberg. Globally consistent 3d mapping with scan matching. *Robotics and Autonomous Systems*, 56(2):130–142, Feb. 2008.

[5] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. *High Accuracy Optical Flow Estimation Based on a Theory for Warping*, pages 25–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[6] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard. Simultaneous localization and mapping: Present, future, and the robust-perception age. *arXiv preprint*, arXiv:1606.05830v2, 2016.

[7] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] F. F. D. Scaramuzza. Visual Odometry: Part I - The First 30 Years and Fundamentals.

[9] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3D mapping with an RGB-D camera. *IEEE Transactions on Robotics*, 30(1):177–187, Feb. 2014.

[10] D. Galvez-Lopez and J. D. Tardos. Real-time loop detection with bags of binary words. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 51 –58, sept. 2011.

[11] D. Galvez-Lopez and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.

[12] L. Goncalves, E. di Bernardo, D. Benson, M. Svedman, J. Ostrowski, N. Karlsson, and P. Pirjanian. A visual front-end for simultaneous localization and mapping. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 44–49, April 2005.

[13] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, winter 2010.

[14] D. Gutiérrez-Gómez. Real-time metric localisation with wearable vision systems, 2016.

[15] D. Gálvez-López. DLib: C++ library with several utilities. [Online; accessed 15-September-2016].

[16] J. Heinly, E. Dunn, and J.-M. Frahm. *Comparative Evaluation of Binary Features*, pages 759–773. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[17] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. *RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments*, pages 477–491. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[18] C. Hertzberg. A framework for sparse, non-linear least squares problems on manifolds, 2008.

[19] B. K. Horn and B. G. Schunck. Determining optical flow. Technical report, Cambridge, MA, USA, 1980.

[20] Y. Hou, H. Zhang, and S. Zhou. Convolutional neural network-based image representation for visual loop closure detection. In *2015 IEEE International Conference on Information and Automation*, pages 2238–2245, Aug 2015.

[21] International Telecomunication Union. Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios. `https://www.itu.int/rec/R-REC-BT.601/`. [Online; accessed 15-September-2016].

[22] M. Jaimez and J. González-Jiménez. Fast visual odometry for 3-d range sensors. *IEEE Transactions on Robotics*, 31(4):809–822, 2015.

[23] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106, Nov 2013.

[24] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, May 2011.

[25] M. Labbé and F. Michaud. Online global loop closure detection for large-scale multi-session graph-based slam. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, Sept 2014.

[26] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[27] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, Feb 2016.

[28] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333–349, 1997.

[29] J. luis Blanco. A tutorial on se(3) transformation parameterizations and on-manifold optimization, 2015.

[30] L. Ma, C. Kerl, J. Stückler, and D. Cremers. Cpa-slam: Consistent plane-model alignment for direct rgb-d slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1285–1291, May 2016.

[31] MRPT.

[32] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015.

[33] P. Newman and K. Ho. Slam-loop closing with visually salient features. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 635–642, April 2005.

[34] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *roceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–652–I–659 Vol.1, June 2004.

[35] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society.

[36] OpenCV. Open source Computer Vision. `http://opencv.org/`. [Online; accessed 15-September-2016].

[37] OpenMP Architecture Review Board. The OpenMP API Specification for Parallel Programming. `http://openmp.org/wp/`. [Online; accessed 15-September-2016].

[38] PCL. Point Cloud Library. `http://pointclouds.org/`. [Online; accessed 15-September-2016].

[39] O. H. D. M. D. K. A. J. D. P. K. J. S. S. H. A. F. Richard A. Newcombe, Shahram Izadi. Kinectfusion: Real-time dense surface mapping and tracking. In *IEEE ISMAR*. IEEE, October 2011.

[40] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[41] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[42] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *3rd International Conference on 3D Digital Imaging and Modeling (3DIM 2001), 28 May - 1 June 2001, Quebec City, Canada*, pages 145–152, 2001.

[43] S. A. Scherer, A. Kloss, and A. Zell. Loop closure detection using depth images. In *2013 European Conference on Mobile Robots (ECMR)*, pages 100–106, Sept 2013.

[44] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[45] H. Shahbazi and H. Zhang. Application of locality sensitive hashing to realtime loop closure detection. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1228–1233, Sept 2011.

[46] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1470–1477, Washington, DC, USA, 2003. IEEE Computer Society.

[47] H. Spies, B. Jähne, and J. L. Barron. Range flow estimation. *Computer Vision and Image Understanding*, 85(3):209–231, 2002.

[48] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proceedings of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[49] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[50] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, Apr 1991.

[51] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. Mcdonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *International Journal of Robotics Research*, 34(4-5):598–626, Apr. 2015.